# Software Reliability – Six Reasons Why it is a Tricky Issue

Reliability claims for software-based systems are fundamentally more difficult than those for other systems:

1. For individual electrical, electronic or mechanical components, reliability claims are usually made on the basis of measured failure rates of a large number of similar components in other installations. For software-based systems, such an approach is not possible because the software will in general be unique for each application.

2. Furthermore, C&I systems are the last to be installed (and generally the last to be designed) in any large construction project. This is worth mentioning because it can mean that, when the Pre-Construction Safety Report (PCSR) (or Safety Analysis Report, SAR) is issued, detailed design of the C&I systems may not be complete. The PCSR or SAR will be based, therefore, on reliability claims for C&I systems which may exist only in concept. An unfortunate consequence of this is that safety analysts, who construct the safety arguments for the PCSR, may make excessive claims for the reliability of C&I systems - so that they can more readily 'achieve' target fault frequencies for major accidents – and the unfortunate C&I designer can then be stuck with implementing unnecessarily high reliability claims for the safety system.

3. Software-based systems attain a complexity that is not achievable for hard-wired systems. It is not unusual for a software-based safety-related distributed control system (DCS) to have several thousand inputs and outputs (I/O). For *hard-wired* systems, each safety function will generally be performed by a small number of components which are separate from other functions. It is therefore generally possible to prove the installed logic absolutely by doing both positive and negative testing, i.e. by testing that the desired combination of inputs always yields the correct output (positive testing) and that all undesired combinations of inputs will not generate any incorrect outputs (negative testing). However, for *software-based* systems with thousands of I/O, where large number of signals may go into the same processor, negative testing becomes impractical because the number of possible combinations of I/O increases to the point where the necessary test time becomes unfeasibly long. Also, software enables algorithms to be implemented in the logic, introducing another level of complexity.

4. Although the focus of attention can sometimes be on application software (which contains the logic and algorithms), operating system software also has safety-significance and should be subject to a similar level of attention and scrutiny. In particular, Windows-based systems are not generally acceptable for high-integrity safety-related applications.

5. Smart sensors and other COTS (Commercial-Off-The-Shelf) components (containing software which is typically for signal processing and A/D conversion) are now ubiquitous. However, manufacturers' concerns about their Intellectual Property Rights (IPR) mean that access to software (for independent review and assessment) can be difficult.

6. Unlike hard-wired systems, you can't physically check how the system is configured. Hence, rigorous software configuration management and change control are absolutely vital to ensure that (a) all software changes retain the same level of integrity as the original software, and (b) that the correct version of software is actually installed on the plant. These controls must also be absolutely rigorous throughout the project lifecycle, i.e. during plant construction, commissioning and operation. The most difficult time is immediately after 'design freeze' (when some design changes will still be occurring) and during commissioning (when faults may be found which necessitate changes).